MPI Version of the Serial Code with One Dimensional Decomposition

Luke Lonergan
High Performance Technologies, Inc.

December 8, 1998

Overview

We will choose one of the two dimensions and subdivide the domain to allow the distribution of the work across a group of distributed memory processors

We will focus on the principles and techniques used to do the MPI work in the model

STEP1: introduce the MPI environment

10 minutes

- cd /mpi_1d/step1
- compile and run the serial code
- edit the file step1.f : (only the main program)
  - include the mpi header file
  - declare integer variables size, rank, ierr
  - initialize MPI
  - determine process group size
  - determine the rank of calling process
  - terminate MPI environment

STEP1: introduce the MPI environment (continued)

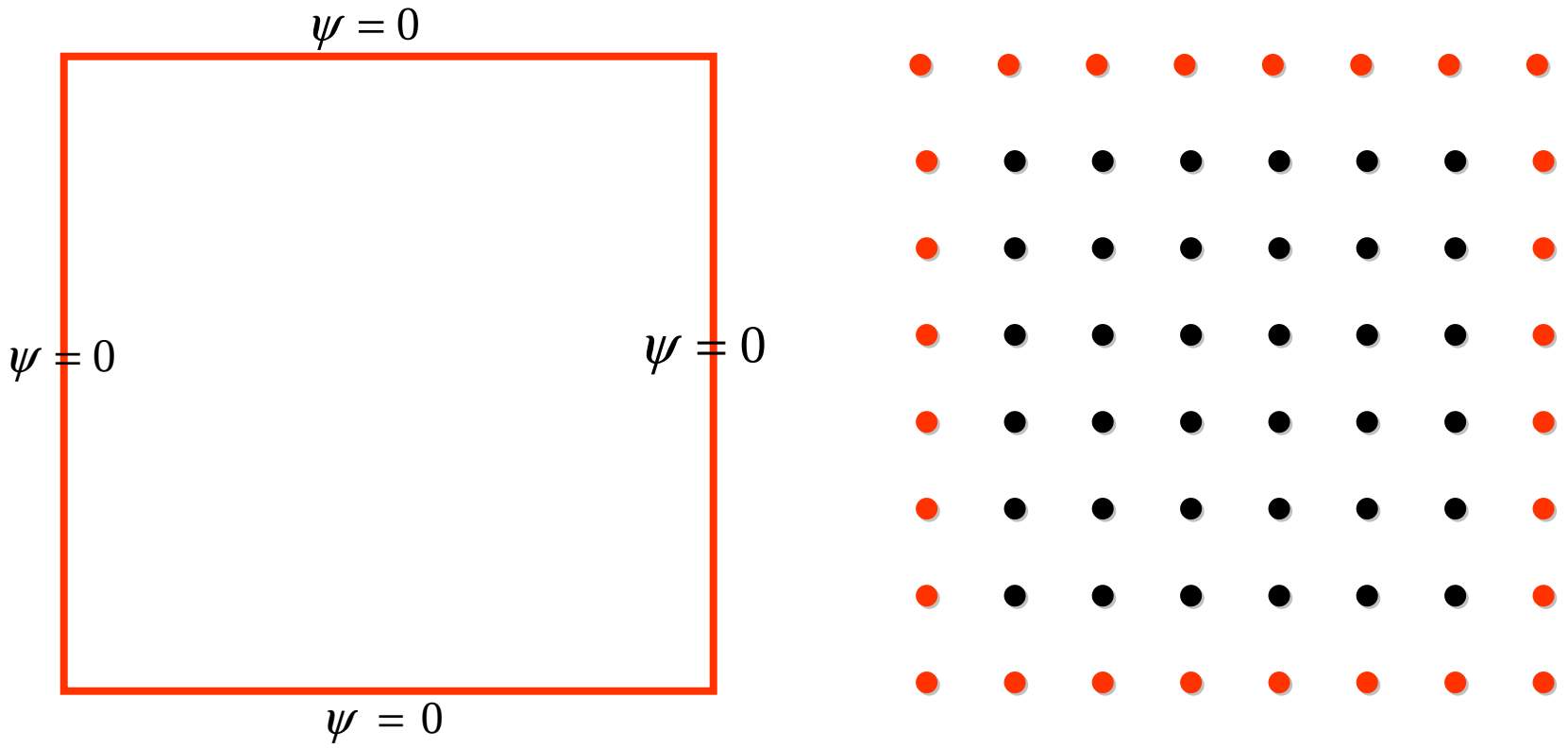
- compile step1.f :

  mpif77 step1.f -o step1.x

- run the executable on one processor

  mpirun -np 1 step1.x

Domain Discretization



Structure of the Serial Code

- namelist : Lx, Ly, α , β, γ
- grid : Δx, Δy
- forcing : $- \alpha \sin(\pi * (j-1) * \Delta y / Ly)$
- initial : $\psi(i, j) = 0.0$ for $i = 1, nx, j = 1, ny$
- bcs : $\psi(i,1) = \psi(i,ny) = \psi(1,j) = \psi(nx,j) = 0$

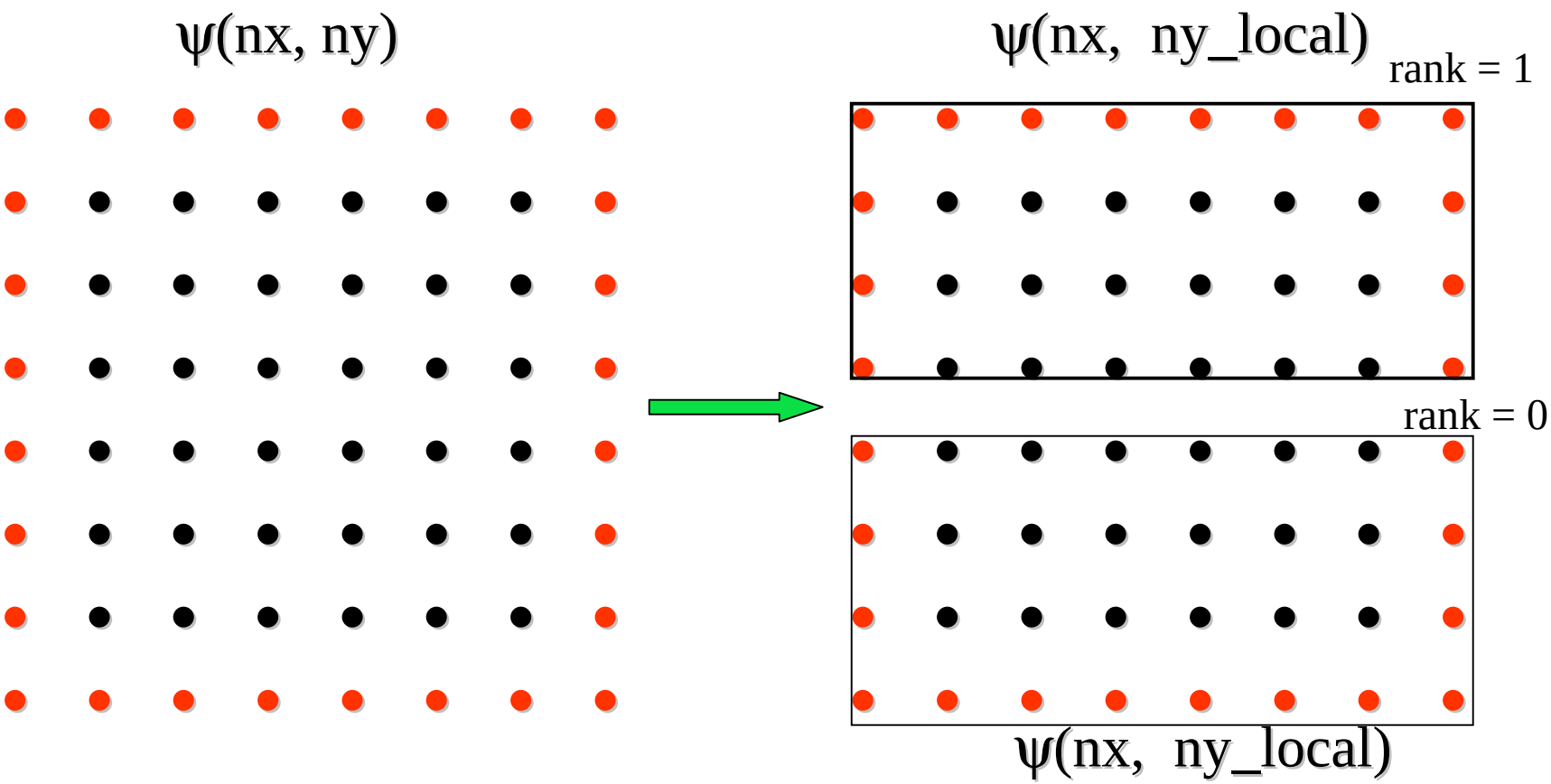
Structure of the Serial Code (continued)

begin iteration: it = 1, itcnt

- jacobi : $\psi\_new(i, j) = a1*\psi(i+1,j) + a2*\psi(i-1,j) + a3*\psi(i, j+1) + a4*\psi(i, j-1) - a5*f(i, j)$; i=2,nx-1,j=2,ny-1
- residual : $(\psi(i,j) - \psi\_new(i, j))**2$; i=2,nx-1,j=2,ny-1
- copy : $\psi(i,j) = \psi\_new(i, j)$
- end iteration

ID Decomposition

Physical domain is sliced into slabs so that computation in each slab will be handled by different processors.

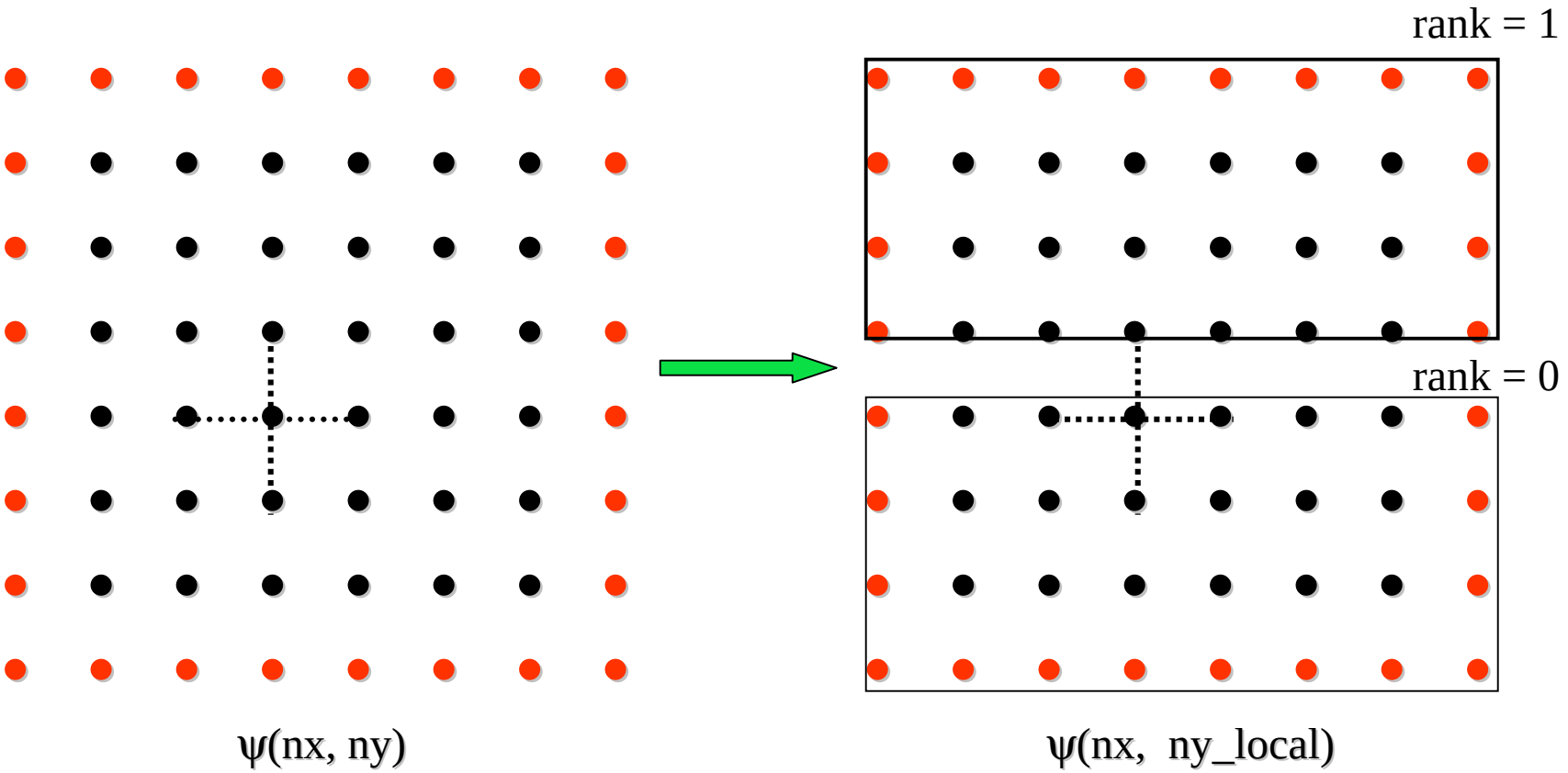


Jacobi Iteration on Decomposed Data

$$\psi\_new(i, j) = a1*\psi(i+1,j) + a2*\psi(i-1,j) + a3*\psi(i, j+1) + a4*\psi(i, j-1) - a5*f(i, j)$$

i=2,nx-1 ; j=2,ny-1

i=2,nx-1 ;j=2,ny_local-1



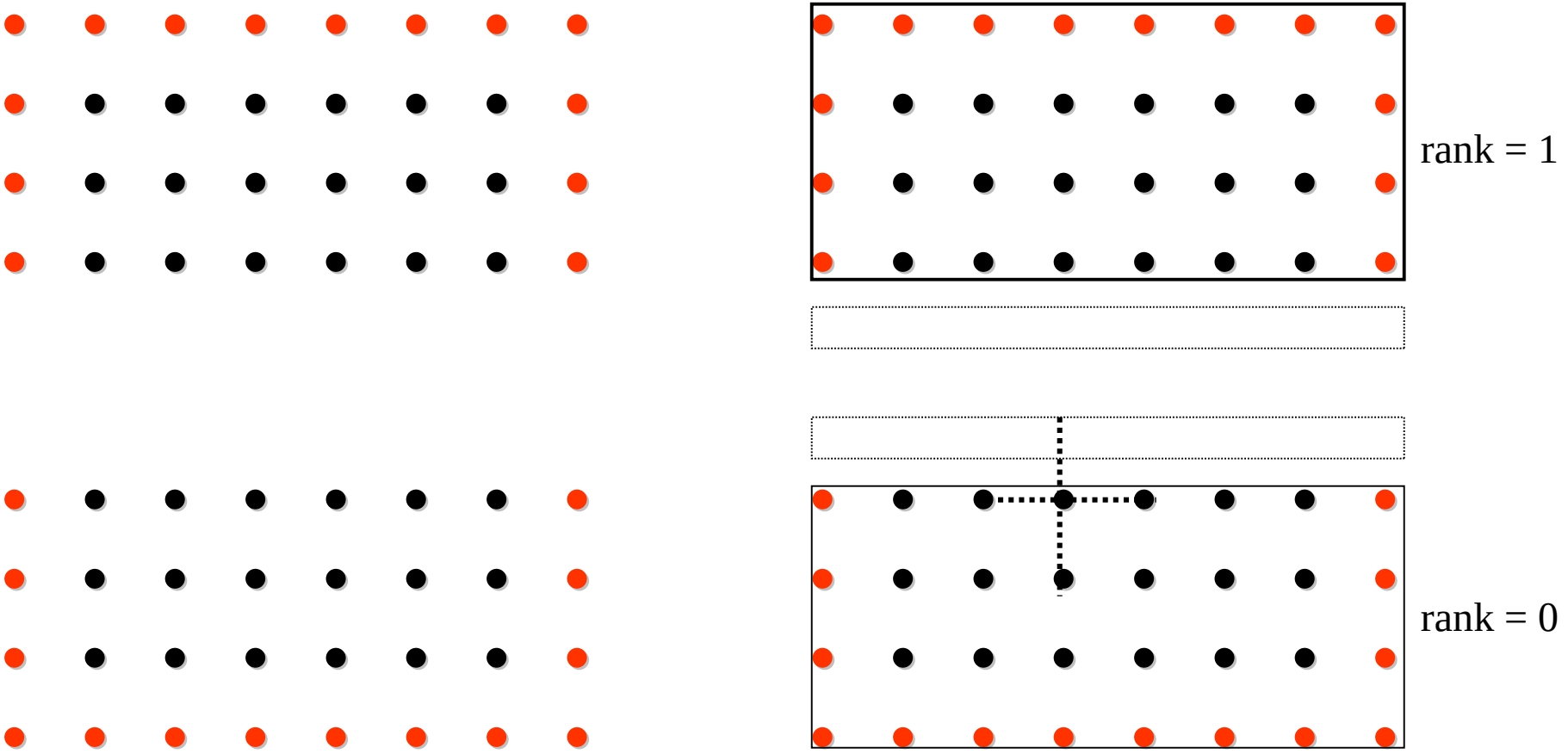
ψ(nx, ny)

ψ(nx, ny_local)

The “ghost” rows

i=2, nx-1 ; j=2, ny-1

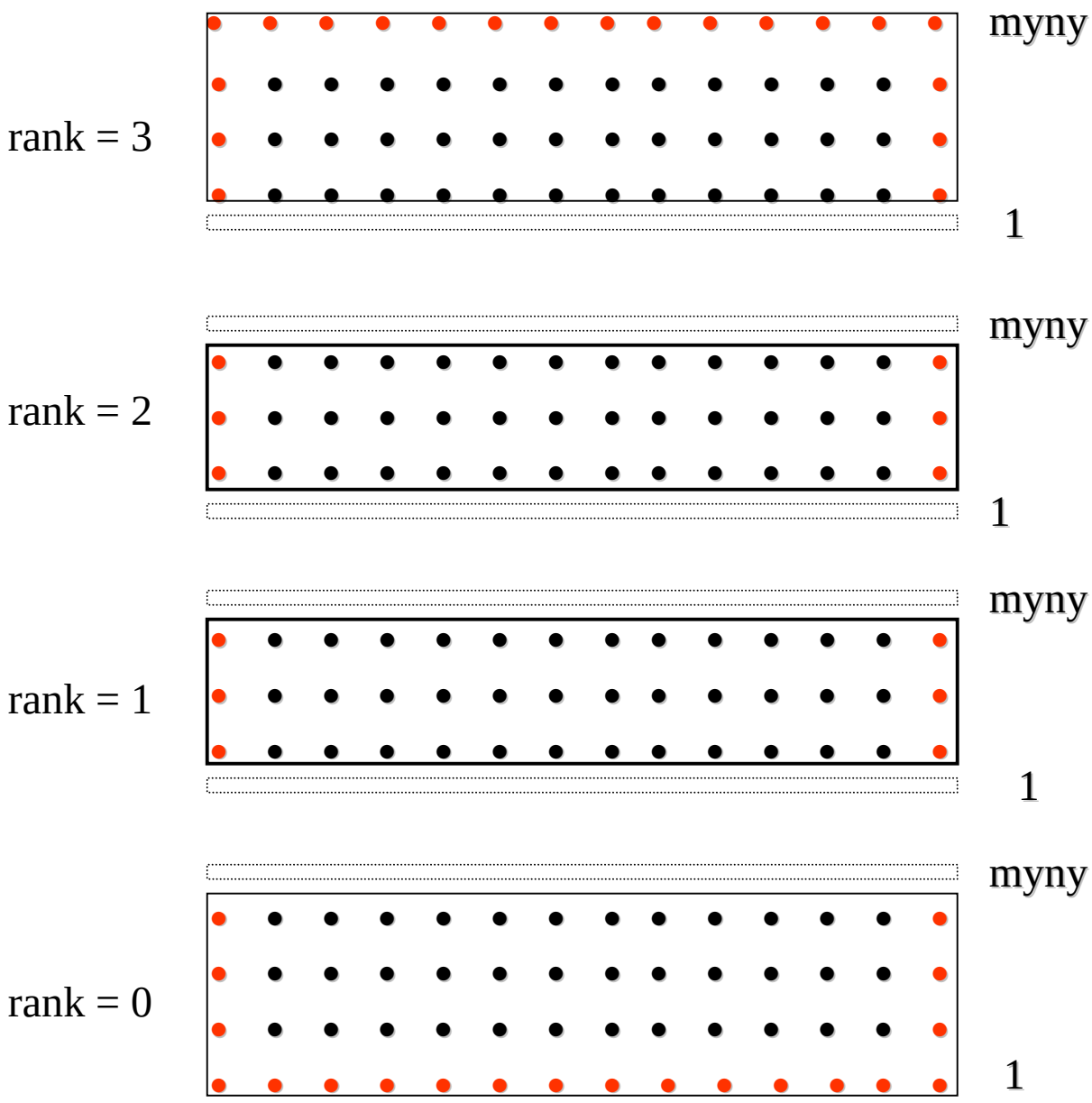
i=2,nx-1 ;j=2, myny-1



ψ(nx, ny)

ψ(nx, myny)

On defining "myny"



One choice:

myny = [(ny-2)/nprocs]

+ 2

STEP2: 1d decomposition of the arrays in the serial code

10 minutes

- cd /mpi1d_step2
- edit step2.f ….. (step2.f = step1.f + comments)

  - declare integer variable nprocs
  - for now set parameter(nprocs=1)
  - declare integer variable myny
  - set parameter(myny = [(ny-2)/nprocs] + 2])

STEP2: 1d decomposition of the arrays in the serial code (continued)

12 minutes

- edit /mpi1d_/step2/ step2.f ….. (continue)
  - change ny to myny everywhere in the code (EXCEPT in subroutine grid and in the parameter statement)
- compile and run step2.f with np = 1

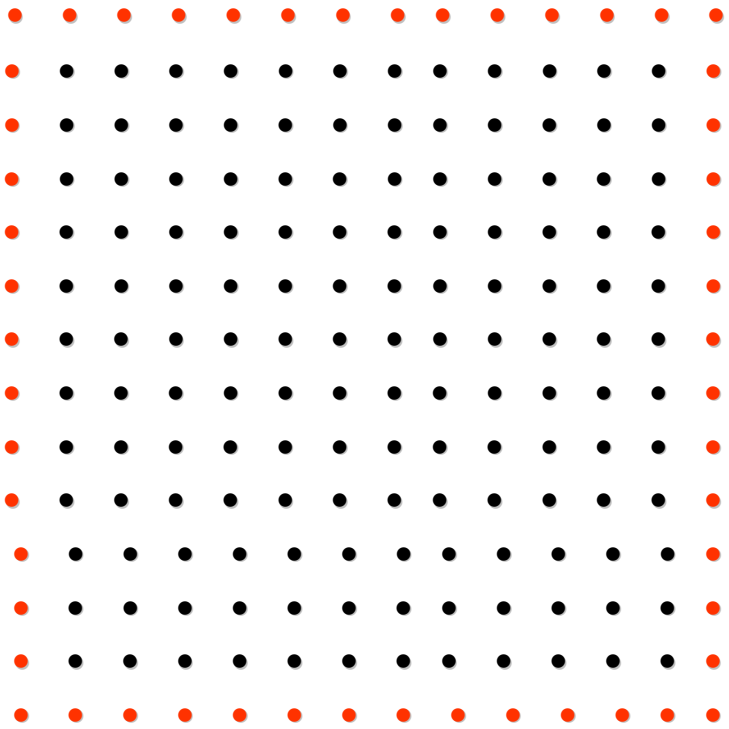
Decomposing the Forcing Function

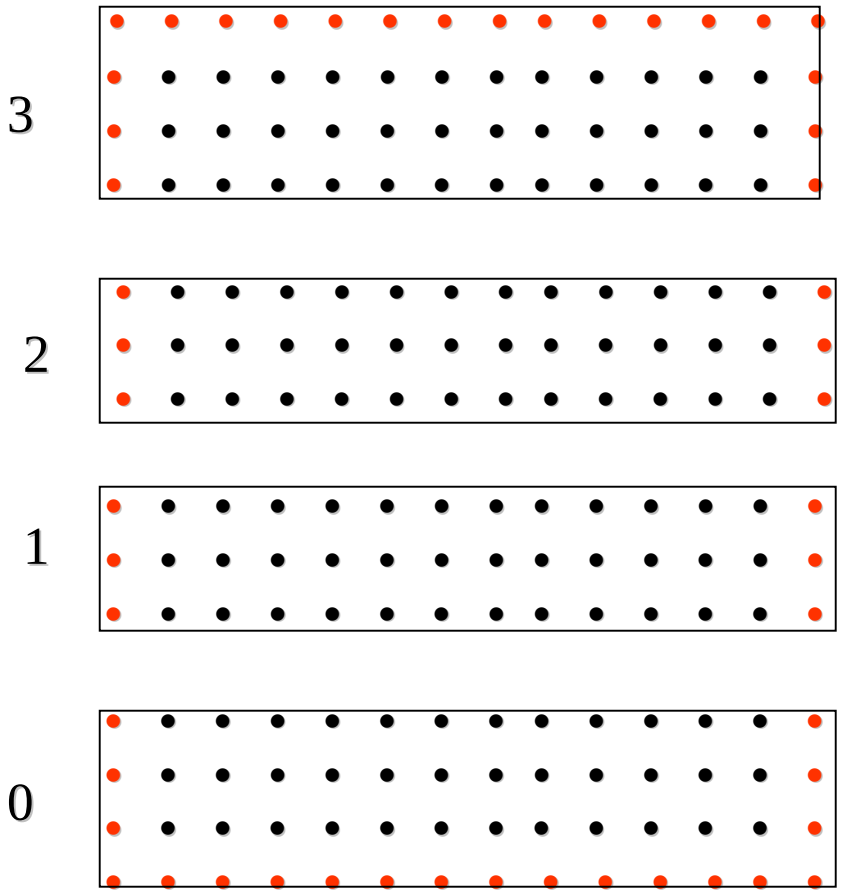
$$f(i,j) = -\alpha \sin(\pi * y(j)/Ly)$$

$yj = (j-1) * \Delta y$

$yj = ystart + (j-1) * \Delta y$

$ystart = rank * (myny-2) * \Delta y$





STEP3: decomposing the forcing function

8 minutes

- cd /mpi_1d/step3
- edit the file step3.f to make the following changes in routine “forcing”:

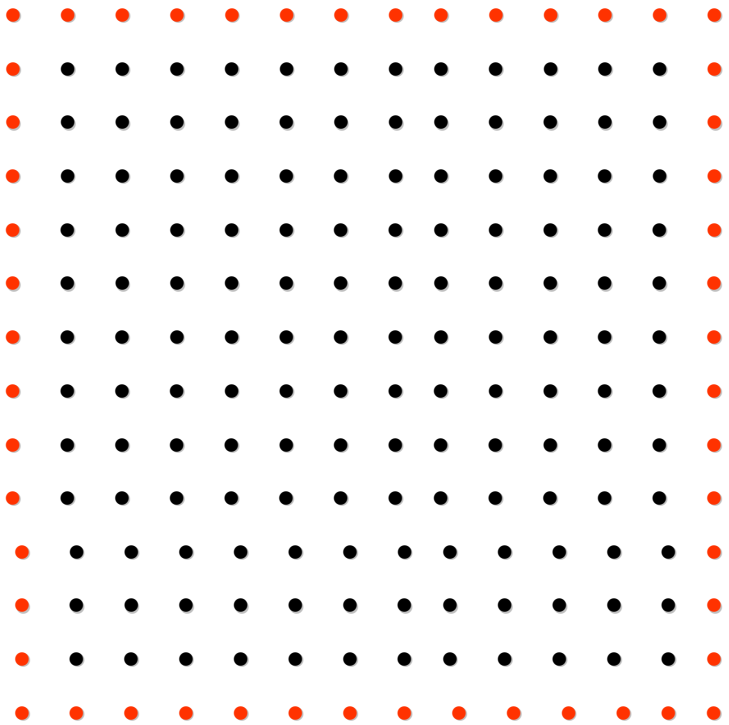
  - include rank as an argument of “forcing”
  - define ystart = rank * (myny-2) * dy
  - replace the statement for yj with

    yj = ystart + (j-1)* dy

- compile and run the code with np = 1

Decomposing the Horizontal Boundary Conditions

$\psi(i,ny) = 0 ; i = 1,nx$

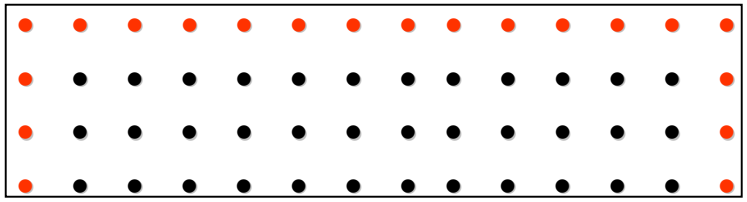


$\psi(i, 1) = 0 ; i = 1,nx$

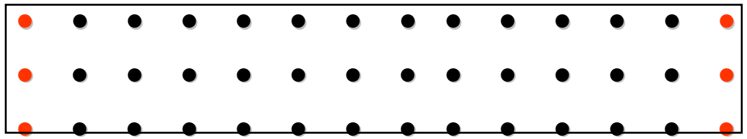
If (rank .eq. size-1)

$\psi(i,ny) = 0 ; i = 1,nx$

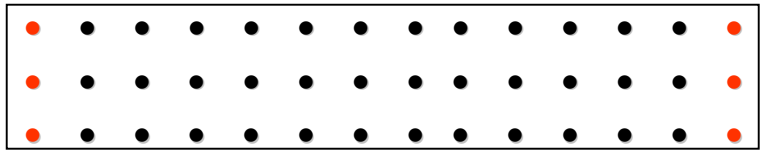
3



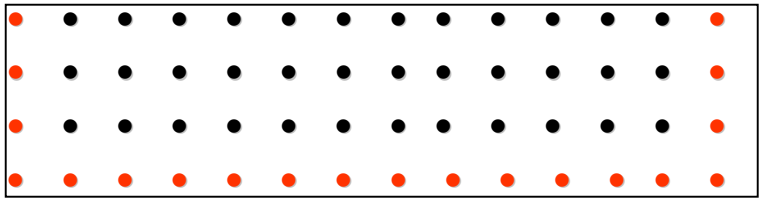
2



1



0



If (rank .eq. 0)

$\psi(i,1) = 0 ; i = 1,nx$

STEP4: Decomposing the Horizontal Boundary conditions

5 minutes

- cd /mpi1d_step4

- edit the file step4.f to make the following changes in the routine “bcs”:

  - include rank and size as arguments of “bcs”

STEP4: Decomposing the Horizontal Boundary conditions

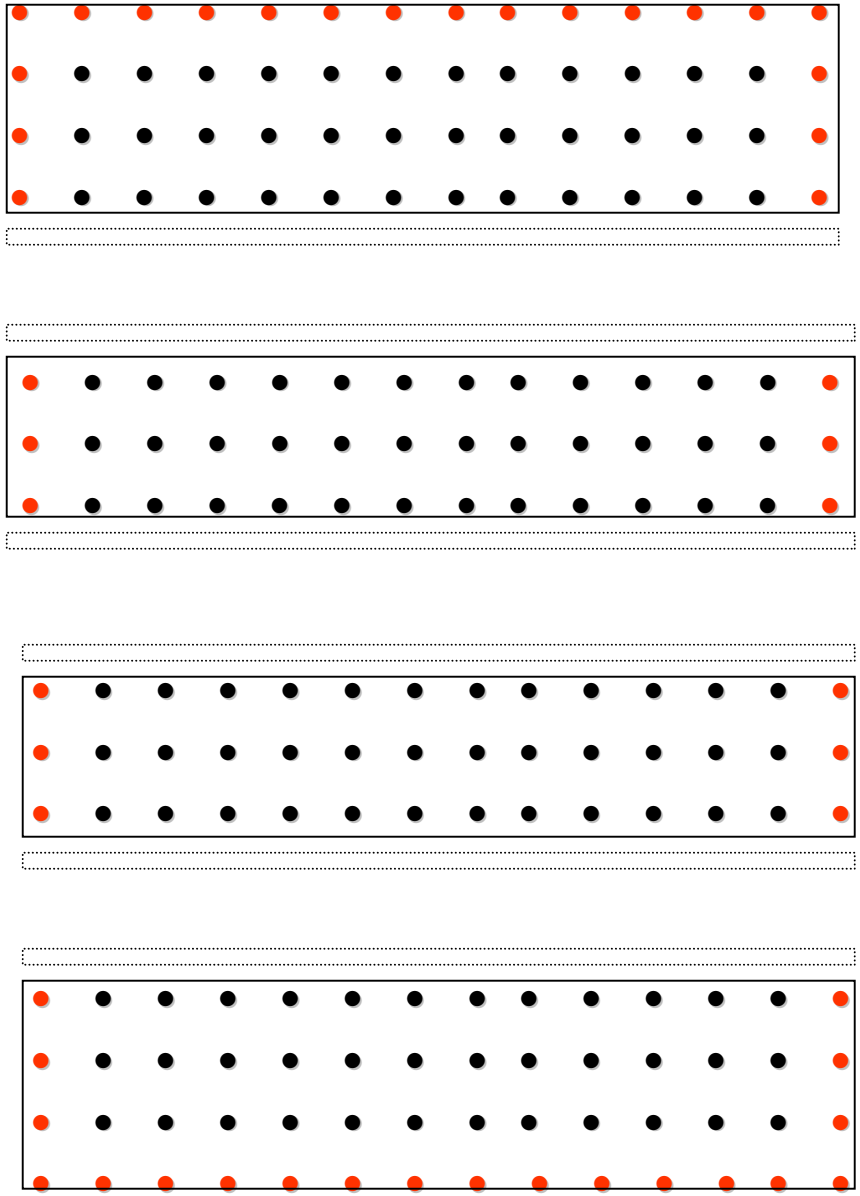
5 minutes

- split the do loop in bcs for x-boundaries into two loops: one for j = 0 and the other for j = myny

- set $\psi(i, 1) = 0$; i = 1, nx only if rank = 0
- set $\psi(i, myny) = 0$; i = 1, nx only if rank = size-1

- compile and run the code with np = 1

Who are my neighbors ?



rank of my neighbor above
=
my_rank+1

rank of my neighbor below
=
my_rank-1

STEP5: finding the ranks of neighbors

3 minutes

- cd /mpi-1d/step5

- edit the file neighbors.f :

  -modify the routine so that it will take rank as input and return integers "up" and "down" where up = rank + 1 & down = rank - 1

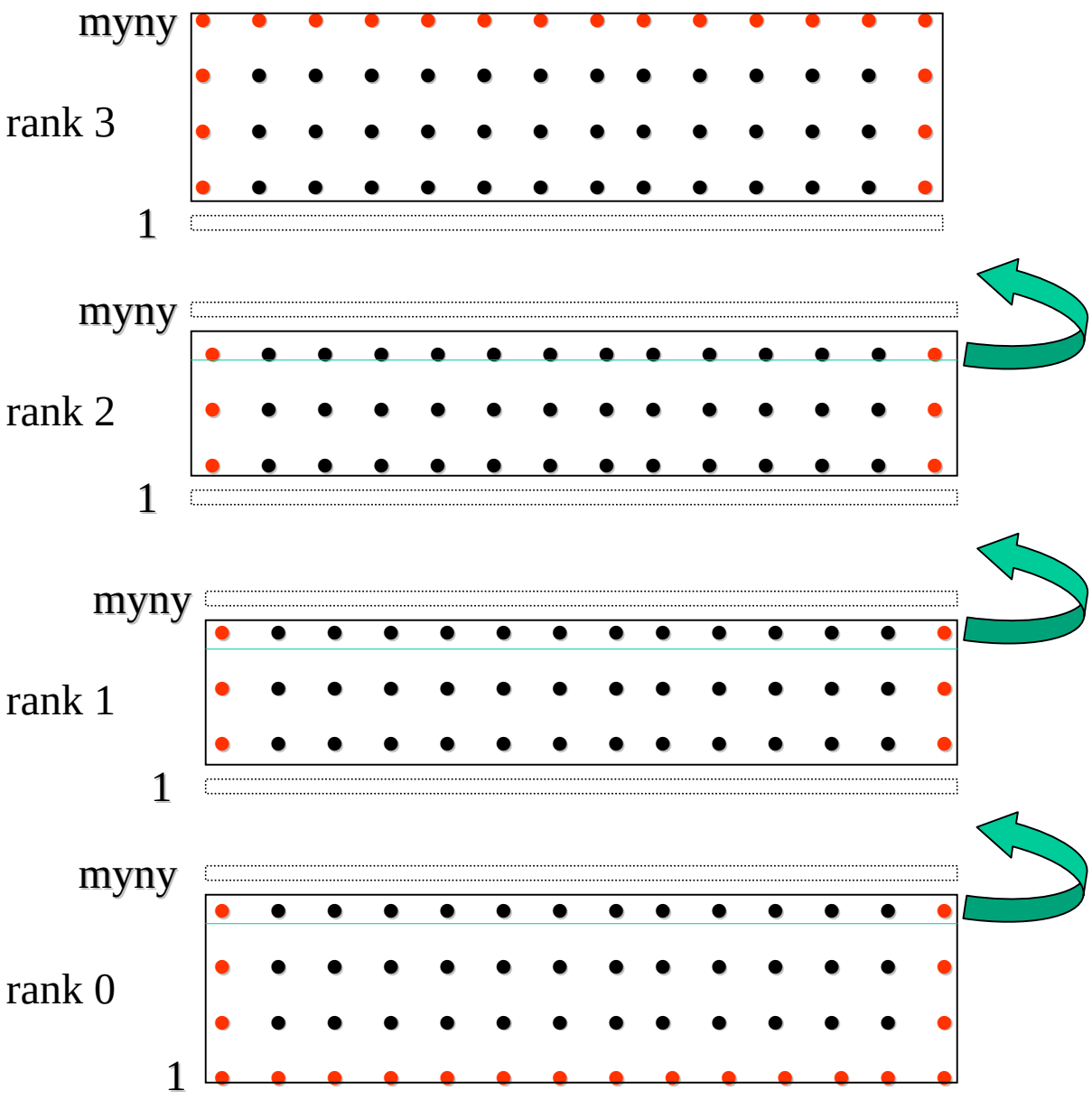
STEP6: Exchange the "Ghost Data" Across the Processors

3 minutes

cd /mpi_1d/step6

- edit the file exchange.f
  - include the mpi header file
  - declare two status arrays: status1 and status2 (integer type, with dimension MPI_STATUS_SIZE)

Exchanging the Ghost Data - (i)



If(rank .lt. size-1)send the last ”interior” row (i.e.ψ(i, myny-1)) to processor above

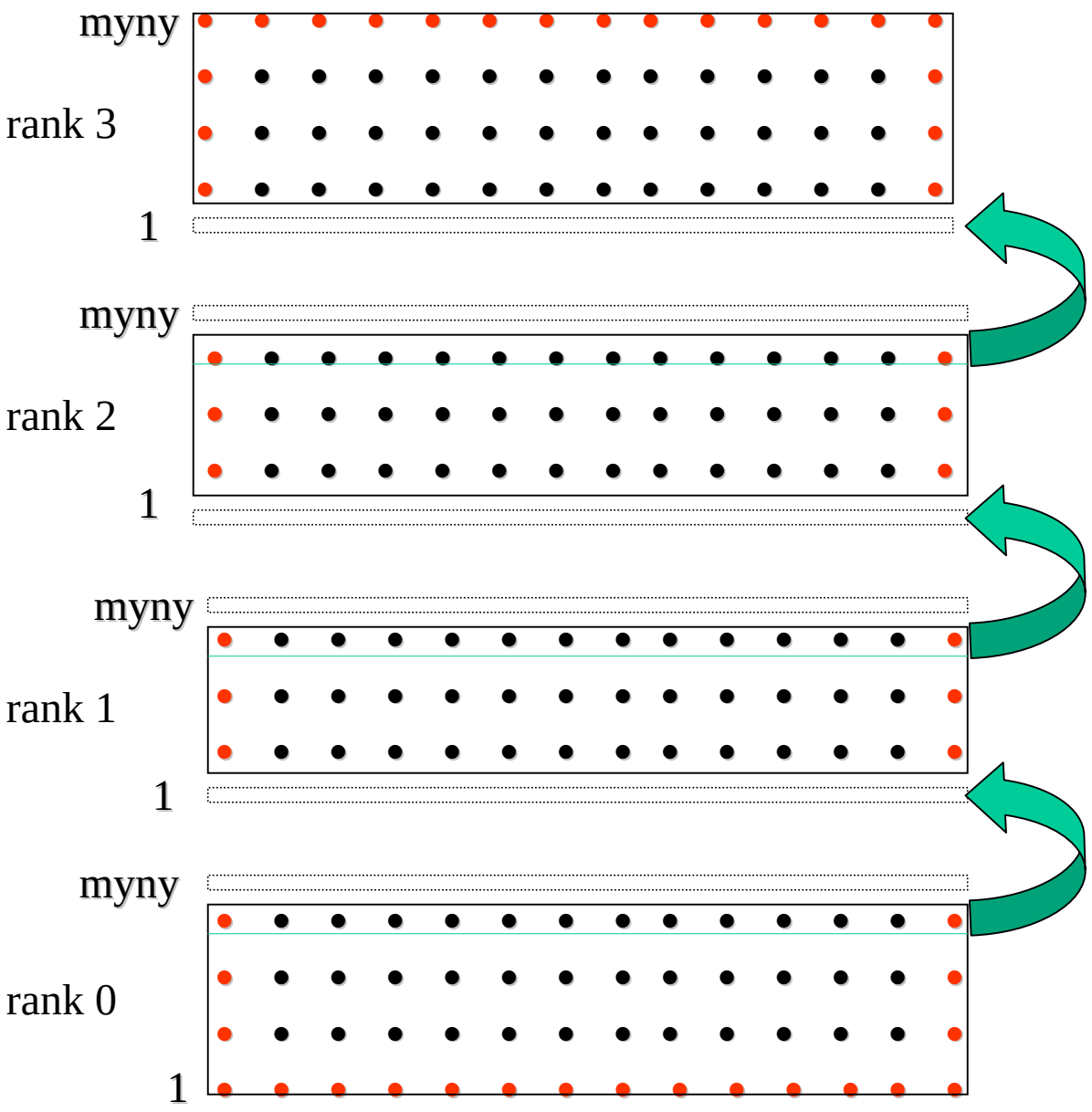
STEP6a: sending (myny-1)th row to the processor above

4 minutes

- cd mpi-1d/step6
- edit the file exchange.f to include the following MPI_SEND call at the indicated location.

```
CALL MPI_SEND(psi(1,myny-1),
                    nx,
                    MPI_REAL,
                    up,
                    999,
                    MPI_COMM_WORLD, ierr)
```

Exchanging the Ghost Data - (ii)



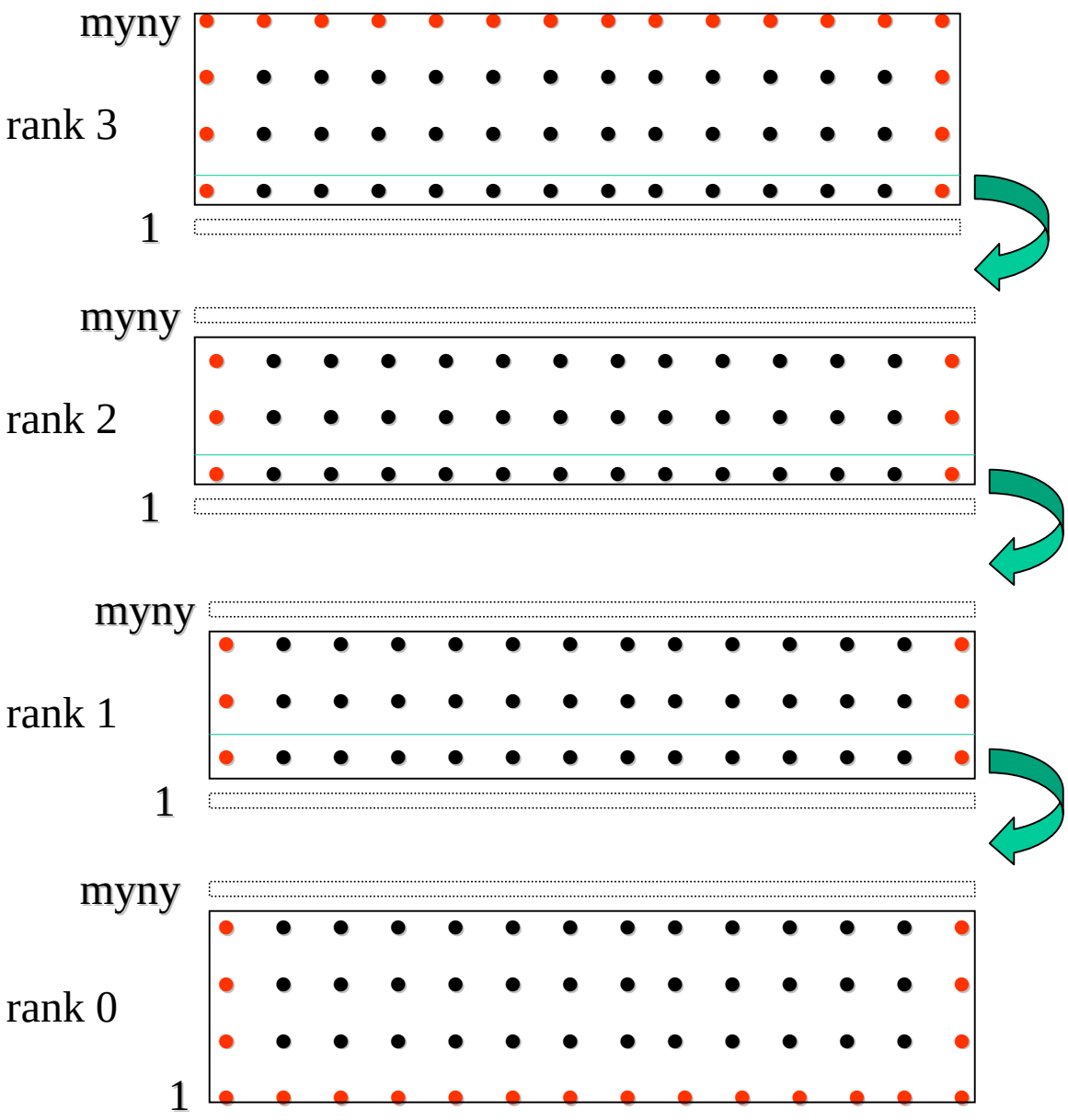
If(rank .gt. 0) receive a row of “real” type elements tagged 999 from the processor below and save it in the location starting at ψ(1,1)

STEP6b: receiving the first "ghost row" from processor below

4 minutes

- cd mpi-1d/step6
- edit the file exchange.f to include the following MPI_RECV call at the indicated location.
- CALL MPI_RECV(psi(1,1),
  nx,
  MPI_REAL,
  down,
  999,
  status1,
  MPI_COMM_WORLD, ierr)

Exchanging the Ghost Data - (iii)



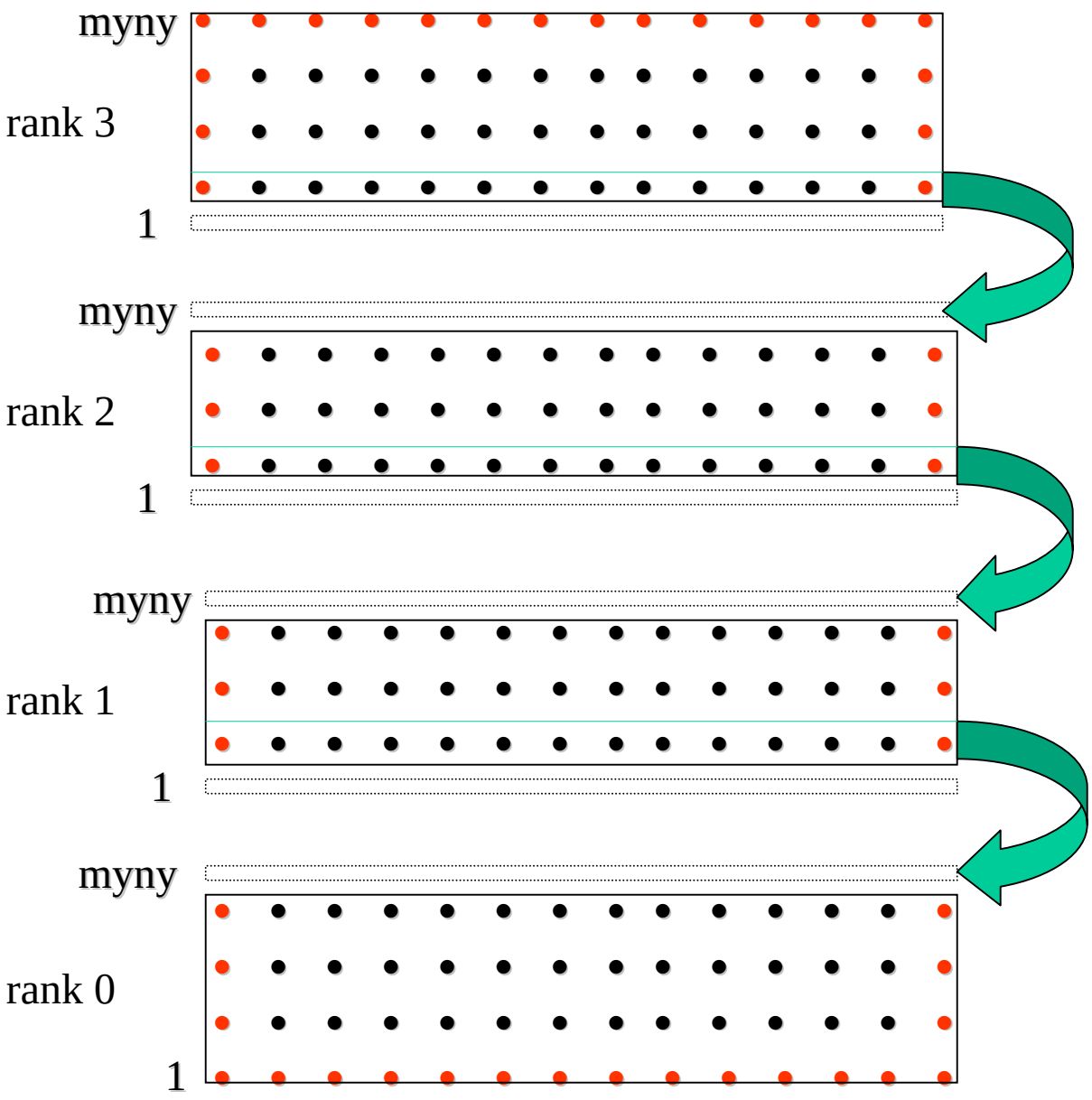
If(rank .gt. 0) send the first ”interior” row (i.e. ψ(i, 2)) to the processor below

STEP6c: sending the second row to the processor below

4 minutes

- cd mpi-1d/step6
- edit the file exchange.f to include the following MPI_SEND call at the indicated location.
- CALL MPI_SEND(psi(1,2),
  nx,
  MPI_REAL,
  down,
  888,
  MPI_COMM_WORLD, ierr)

Exchanging the Ghost Data - (iv)



If(rank .lt. Size-1) receive a row of "real" type elements tagged 888 from the processor above and save it in the location starting at ψ(1,myny)

STEP6d: receiving the last "ghost row" from processor above

4 minutes

- cd mpi-1d/step6

- edit the file exchange.f to include the following MPI_RECV call at the indicated location.

STEP6d: receiving the last "ghost row" from processor above (continued)

- CALL MPI_RECV(psi(1,myny),
  nx,
  MPI_REAL,
  down,
  888,
  status2,
  MPI_COMM_WORLD, ierr)

STEP6: a final check

1 minute

- cd /mpi-1d/step6
- compare your exchange.f with
  /mpi_1d/step6/completed/exchange.f

STEP7: Append neighbors.f and exchange.f to step4.f

4 minutes

- cd /mpi_1d/step7
- edit the file step7.f

  (step7.f = step4.f + neighbors.f + exchange.f +

  + additional comments)

  - declare integers up & below in the main program

  - insert a call to “neighbors” just before starting the iteration.

STEP7: Append neighbors.f and exchange.f to step4.f (continued)

4 minutes

- cd /mpi_1d/step7
- - insert a call to “exchange” inside the iteration loop - just before calling jacobi
- compile and run your code, but still with np = 1

STEP8: collecting the local residuals and reducing it to a global value

4 minutes

- cd /mpi_1d/step8

- edit the file step8.f to make the following changes in the routine “residual”

  - include the mpi header file
  - declare a new real variable lnorm: for local norm
  - replace gnorm with lnorm in the rest of the routine EXCEPT in the sqrt statement

STEP8: collecting the local residuals and reducing it to a global value (continued)

8 minutes

- use MPI_ALLREDUCE to collect the local residuals, to reduce it to a global value through a pre-defined operation and to distribute it back to all processors.

STEP8: collecting the local residuals and reducing it to a global value (continued)

- CALL MPI_ALLREDUCE(local buffer (lnorm),
  global value to be received (gnorm),
  # of elements in local buffer (1) ,
  datatype (MPI_REAL),
  reduction operation (MPI_SUM),
  MPI_COMM_WORLD, ierr)

- insert the call in residual - before computing sqrt
- compile and run the code, but still with np = 1

Step9 : The (Almost) Final Step

8 minutes

- cd /mpi_1d/step9

- edit step9.f to make the following changes in the main program
  - set nprocs = 4 (finally !!)
  - modify the print statement so that only processor 0 prints the output

- compile and run the code with np = 4

Residuals from the serial code and the mpi_1d version- What happened here ?

it_cnt	Serial Code	mpi_1d
1	1169699.38	1169729.5
2	1167068.75	1167098.12
3	1165272.38	1165305.12
4	1163841.5	1163839.38
5	1162584.12	1162584.12
6	1161434.	1161446.88
7	1160383.62	1160407.75
8	1159442.88	1159423.75
9	1158526.38	1158516.88
10	1157625.25	1157664.

Residuals from the serial code and the mpi_1d version- What happened here ?

Hint: sensitivity to the order of summation

STEP10:The Finishing Touch

4 minutes

- cd /mpi_1d/step10
- edit stml_serial.f to make the following changes in the routine residual:

-comment out gnorm=gnom+[ψ(i,j)-ψ_new(i,j)]**2

- insert gnorm=max (gnorm, abs(ψ(i,j)-ψ_new(i,j)))

- comment out gnorm = sqrt(gnorm)

- compile and run this modified version of stml_serial.f

STEP10: The Finishing Touch (continued)

- 4 minutes
- edit the file step10.f to make the following changes in the routine “residual
  - -comment out the statement for lnorm
  - - replace it with

    lnorm = max(lnorm, abs(ψ(i,j)-ψ_new(i,j)))
  - - comment out gnorm = sqrt(gnorm)

STEP10: A Finishing Touch (continued)

- 2 minutes

  - compile and run the code with np = 4

  - How does the results compare with that of the modified stml_serial.f

- You can now rename step10.f to stml_mpi1d.f